ISSN 2224-087X. Електроніка та інформаційні технології. 2024. Випуск 27. С. 3–24
Electronics and information technologies. 2024. Issue 27. P. 3–24

UDK 004 DOI: https://doi.org/10.30970/eli.27.1

SELECTED ASPECTS OF DIGITAL REPRESENTATION OF INFORMATION SYSTEMS

A. Shuparskyy, Y. Furgala

Ivan Franko National University of Lviv,
50 Drahomanova St., 79005 Lviv, Ukraine
anatoliy.shuparskyy@lnu.edu.ua, yuriy.furhala@lnu.edu.ua

From the viewpoint of the evolution of computing devices and corresponding use cases, the article structures an information systems timeline progressing through early electromechanical devices, electronic vacuum tubes, solid-state and integrated circuit electronics, the advent of microprocessors, personal and remote computing, and distributed systems of numerous autonomous devices, while exploring different approaches to a digital representation of computational entities. By establishing the research scope and organizing scholarly sources chronologically, the article reveals and selects connections between individual events, provides an overview and critical analysis, and highlights expected future expansions and shifts in approaches to digital representation.

Thus, the article examines a shift from operations and operands, their further complication into code and data structures, and the transition from procedural to structured and object-oriented programming (OOP). Client-server applications implemented with static OOP and relational data management are examined as the pinnacle of monolithic architecture. The subsequent domain-driven design (DDD) and microservices architecture are examined as contemporary methods for remote cloud computing environments. The article then discusses the rise of the Internet of Things (IoT), the emergence of smart things and digital twins, describes advanced and novel use cases of global digitalization, such as Industry 5.0 ideas, and reveals the limitations of extant methods for corresponding digital representation.

Ultimately, the article introduces a novel method for digital representation employing the post-non-classical paradigm in computer science, which eschews predefined structures in favor of dynamic, interaction-based representations, enabling flexible and adaptive design of distributed systems. Future research directions include the formal specification of this approach and the development of tools for its implementation in complex distributed systems.

Keywords: digital representation, solution architecture, programming paradigms, distributed systems, internet of things, digital twin, industry 5.0

Introduction

At the end of 1913, a continuous car assembly system was launched at Henry Ford's factory in Highland Park, Michigan, USA. The entire assembling process was divided into simple, short operations, each performed by a separate worker. The assembly time for a single car was reduced from more than 12 hours to just 93 minutes, and eventually, the factory produced a new car every 24 seconds. By 1927, more than 15 million Ford Model Ts had been manufactured [1, 2].

Specialization, distribution of functionality, and simultaneous execution of different parts of a common task marked the advent of the Second Industrial Revolution. As we now enter the era of the Fifth Industrial Revolution, Industry 5.0, it is worth considering the development of

information technologies from the viewpoint of the formation of complex distributed systems, particularly the methods and approaches of representing digital entities within the sequence of evolution and the context of use cases.

Nouns and Verbs

The progression of computing devices from mechanical calculators to early semiconductor computers [29, 30] marked a transition from simple arithmetic operations to complex sequences of commands, all based on representing data as numbers.

Originating with mechanical calculators, the first generation of computers included mechanical and electromechanical devices. The Pascaline [3], Blaise Pascal's arithmetic machine invented in 1642, was a pioneering mechanical calculator capable of addition and subtraction. Gottfried Wilhelm Leibniz's arithmetic machine, the Stepped Reckoner [4], developed in 1672 could perform multiplication and division. Charles Babbage's Difference Engine [6], an advanced mechanical calculator developed in the 1830s, could approximate logarithms and trigonometric functions using polynomials. Babbage's later invention, the Analytical Engine [6], further advanced computational concepts, although it was never completed.

In 1941, German engineer and computer pioneer Konrad Zuse developed the Z3 computing machine [7] utilizing electromechanical relays, making it the world's first programmable computer. It was designed for use in aircraft and early rocket construction and calculations at the German Aerodynamic Research Institute. The Z3's upgraded version, the Z4 [8], completed by Zuse in 1950 after World War II, became the first commercially sold computer. It was acquired by the Swiss Federal Institute of Technology in Zurich and later by the Franco-German Institute of Research in France. In 1944, under the direction of American computing pioneer Howard Aiken, IBM constructed the Harvard Mark I, also known as the Automatic Sequence Controlled Calculator (ASCC) [9], for Harvard University. In a sense, it marked the completion of Babbage's Analytical Engine. The US Navy employed this advanced electromechanical relay-based programmable calculator for ballistic calculations and the study of the implosion of the first atomic bomb at the end of World War II.

The second generation of computers utilized electronic vacuum tube technology. The first computational systems of this generation, Colossus Mk I and Mk II, developed by engineer Thomas Harold Flowers with the British General Post Office in 1943, were employed for the decryption of intercepted radiotelegraph messages from the German Lorenz SZ 42 cipher machine, which was more advanced than the Enigma [10]. Subsequently, in the UK, for use in the British nuclear program, the Manchester Mark 1 computing system, which was the first to utilize drum memory, was developed at the Victoria University of Manchester in 1948 as the successor to the Manchester Baby computing experiment, which was the first to utilize electronic vacuum tube memory [11, 12]. Their successor, the Ferranti Mark 1, built in 1951, became the first commercially sold computer utilizing vacuum tube technology [13, 14].

The well-known ENIAC computing system [15, 16], developed in 1945 by John Presper Eckert and John Mauchly at the University of Pennsylvania, USA, was the first programmable electronic universal digital computer, performing calculations for the US nuclear program and weather modeling. The EDVAC computing system [17, 18], developed in 1949, was employed at the Ballistic Research Laboratory and the US Army's DEVCOM Research Laboratories. The first unit of the UNIVAC I computing system [19] was employed at the US Census Bureau in 1951. A total of 46 UNIVAC I systems were sold to various federal agencies and companies for economic and statistical tasks and even once for predicting the outcome of the US presidential election.

The Whirlwind I computing system [20], developed in 1951 for the US Navy at the MIT Servomechanism Laboratory, was used for early digital air defense and air traffic control operations and featured the first graphical display with a light pen for targeting. IBM's first commercial vacuum tube computer, the IBM 701 [21, 22], also known as the “Defense Calculator,” was developed in 1952 to assist the US during the Korean War. All IBM 701 units were employed [23] at government organizations, military establishments, and aircraft manufacturers on the US West Coast. The IBM 701 was the first computer for which software was created for translating into English from another language.

The IBM 704 mainframe [24], developed in 1954, was the first mass-produced computer with hardware for floating-point operations, establishing the subclass of "scientific architecture" for subsequent IBM computers. The FORTRAN and LISP programming languages were first developed for use for the IBM 704 [25, 38]. In 1957, John McCarthy, an American computer scientist [26], the inventor of the LISP language and the first to use the term "artificial intelligence" in 1955, began the first implementation of a time-sharing system on a modified IBM 704. In 1962, Bell Labs used the IBM 704 for the first human speech synthesis.

At the Kyiv Institute of Electrical Engineering of the Ukrainian SSR, under the direction of Serhiy Lebedev, the Small Electronic Calculating Machine (MESM) was developed between 1948 and 1950, utilizing about 6000 vacuum tubes [27, 28] Deployed in Feofaniya in the Kyiv region, MESM is noted as the first programmable computer using vacuum tubes in continental Europe. MESM was used for scientific, industrial, and military applications.

The third generation of computers utilized solid-state discrete electronics and integrated circuit technology. The invention of the transistor by John Bardeen, Walter Brattain, and William Shockley in 1947 [31] marked a revolutionary advance in computing technology, replacing bulky vacuum tubes with smaller and more reliable components. The IBM 608 programmable calculator [32], introduced in 1957, was the first commercial computer to be composed solely of transistors. The D17B onboard computer for the US Minuteman-1 intercontinental ballistic missile, developed in 1962 by North American Aviation's Autonetics division [33], was among the earliest instances of a computer utilizing transistor technology.

A significant milestone in the development of computing systems based on discrete electronics and integrated circuits was the American manned lunar flight program Apollo. The Apollo missions employed two principal computing systems: the onboard Apollo Guidance Computer (AGC) [34] and the IBM System/360 [35] mainframe on the ground. The AGC, developed by MIT in the early 1960s and first tested in 1966, was one of the first computers utilizing integrated circuits. The AGC provided real-time space flight control and navigation for the missions.

The IBM System/360 Model 91 mainframe [36], deployed at NASA's Goddard Space Flight Center as a ground-based flight computing system, was based on discrete bipolar transistor and diode technology with hybrid module packaging. The IBM System/360 Model 91 introduced one of the first implementations of a Complex Instruction Set Computer (CISC) computation core architecture using microcode for executing complex commands, with support for out-of-order execution and instruction pipelining. The procedural programming language PL/I (Programming Language One), which became the prototype for PL/M (Programming Language for Microcomputers) used in early microprocessors and later the DOS (Disk Operating System) family for early portable computers, was first developed as part of the IBM System/360 family.

The AGC's DSKY data input-output interface in the Apollo mission's astronaut module, with its calculator-like keyboard, was a clear symbol of the early era of computing devices in approaches to data representation and manipulation. The commands were entered numerically

using a "verb-noun" system, where verbs described the operations and nouns represented the data to which the operations applied. The structures of operations and data in the computer's memory defined the main computational models of the time.

This approach directed scientific and engineering thinking toward the active development of discrete mathematics, numerical methods in algebra, and formal methods in programming. Mathematical logic, set theory, algorithm theory, automata theory, and algorithms for numerical and iterative methods for solving linear and nonlinear, integral and differential equations were rapidly developed. These, along with procedural programming languages such as FORTRAN [25] (developed by IBM in the early 1950s for numerical methods and scientific calculations), ALGOL [37] (developed in the mid-1950s by a team of European and American computer scientists at the Swiss Federal Institute of Technology in Zurich as an imperative algorithmic language), LISP [28] (developed at MIT in the late 1950s as a procedural-functional language for mathematical notation), and COBOL [39] (developed within US Department of Defense initiatives in the early 1960s for business and management tasks), became the principal methods and tools for performing computing tasks.

Considering the mentioned use cases, the first-to-third generation computers were employed to address scientific and engineering tasks primarily involving computation operations in mathematical analysis, aerodynamics, hydrodynamics, ballistics, space navigation, meteorology, geophysics, chemical processes, nuclear processes, material strength and shape, crystallography, among others. They also initially addressed statistics, economics, banking, and accounting tasks for governments and corporations.

The early computing systems were developed in an environment defined by individual, unique, and innovative authorial solutions, often under strict and secretive regulation by governments and corporations. The computing science pioneers were qualified scientists, teachers, and graduate students from world-leading educational and scientific institutions. The creation of computer programs satisfying widespread use cases was primarily a scientific endeavor. The prevailing methods of the classical scientific paradigm prompted consistent, predictable development processes, with hierarchical management often based on the individual characteristics of engineers and program directors.

The unchanging demand for solutions to increasingly complex and diverse computing tasks highlighted a fundamental question in the field of information technology: how to ensure the growth of computing systems' productivity? Technological progress in the creation of computer components partially answered this question, but a boost in clock speed alone did not necessarily determine overall productivity. The complexity of computing systems required the effective use of hardware, software, and human resources for maintenance, program development and deployment, data preparation, and a series of related computations. The lack of coordinated and integrated functioning of these necessary resources could result in delays in the outcome.

The optimal approach to ensuring the effective utilization of resources was through the compartmentalization and specialization of the various components and processes of a system into distinct layers and blocks. For instance, this entailed the specialization of hardware and software architecture, such as types of registers and memory, types of numerical data, the separation of microcode in the computing core from operating system code and user program code, etc. Among other aspects, this involved code blocks and procedures developed by diverse teams intended for reutilization in various tasks, and the emergence of early high-level computer languages and procedural compilers that simplified programming (in comparison to machine code) by enabling abstraction levels of data and program representation.

The concept of separate terminals with the ability to receive various tasks from each emerged to optimize the working and idle time of computing blocks between individual computation episodes. Starting in the 1950s, mainframes began to support batch processing, which permitted the sequential execution of tasks from different terminals as soon as the system had available time. The FORTRAN Monitor System (FMS) operating system was designed to support such batch processing. In the 1960s, IBM standardized Job Control Language (JCL) [40], which subsequently supported the execution of batch tasks from remote terminals.

In 1961, the Massachusetts Institute of Technology (MIT) developed the first known time-sharing operating system [41], the Compatible Time-Sharing System (CTSS). As the name implies, CTSS maintained backward compatibility with FMS. Subsequently, in 1965, the inaugural email program for the CTSS operating system was developed at MIT, enabling users to exchange messages via remote terminals connected to a telephone line. In 1962, the BBN company, originating from MIT, developed the BBN Time-Sharing System [42] based on the DEC PDP-1 computer system. In 1964, Dartmouth College developed the Dartmouth Time-Sharing System (DTSS) [43], based on General Electric's GE-225 computing and DATANET-30 interface systems. Initially, 20, and later 40, remote teletype terminals were connected to the DTSS. For this system, a team of Dartmouth College students led by professors John Kemeny and Thomas Kurtz developed the compiler for the new BASIC computer language (Dartmouth BASIC) [44], to teach programming to beginners. The remote terminals thus became the first conceptual prototypes, laying the foundation in the mid-1960s for the first defense and scientific computer network, ARPAnet [45, 46], which became a prototype of the Internet.

Amid Stack and Queue

The substantial growth in computer performance during the late 1950s and early 1960s led to an increase in the complexity of computer programs. Consequently, the difficulty of maintaining them using procedural programming became apparent, prompting further search for approaches to optimize programming. One such approach involved enhancing the structural division of programs, levels of access to data and procedures, etc. As a result, high-level programming languages gained more algorithmic expressiveness and structure. For example, the ALGOL language, adopted by engineers in Europe and North America, advanced the structural organization of procedural languages [37]. The subsequent evolution of this approach established the foundation for structured programming and formed the basis of the Pascal and C languages that were introduced in the 1970s.

Developing programs to control early graphical user interfaces (GUIs) became particularly challenging using procedural programming. Existing languages did not provide simple ways to represent geometric elements using numerical structures and nested procedures, nor could they effectively program GUI interactions with the user. Reflecting on this challenge, in 1963, American computer scientist Ivan Sutherland presented Sketchpad in his doctoral dissertation [47]. Sketchpad was a graphical user interface program that enabled the manipulation of graphical shapes on a cathode-ray tube using a light pen, and in which the terms "objects" and "instances" were first conceptually used.

In 1962, Norwegian scientists Kristen Nygaard and Ole-Johan Dahl studied the modeling systems of interacting elements. They found that language structures based on stacks were not an effective representation of such systems. Instead, they determined that language structures based on queues were needed, which ALGOL did not provide. Their new programming language [48], SIMULA (and later SIMULA 67), became the first language to conceptually use the terms "class/subclass" and its “type.” Nygaard noted:

“... SIMULA' SIMUlation LAnguage' represents an effort to meet this need with regard to discrete-event networks, that is, where the flow may be thought of as being composed of discrete units demanding service at discrete service elements, and entering and leaving the elements at definite moments of time. Examples of such systems are ticket counter systems, production lines, production in development programs, neuron systems, and concurrent processing of programs on computers.” [49]

American computer scientist Alan Kay, while studying for his Ph.D. in computer science at the University of Utah from 1966 to 1969, was inspired by the ideas of Sketchpad and SIMULA, as well as the influence of MIT's artificial intelligence laboratory pioneers who conceptualized LISP atoms with attributes as a kind of "objects." From a use case perspective, he was among the first to understand the true implications of Moore's Law [50] for the computer industry, predicting the advent of portable computers (due to the rapid miniaturization of components) and their interactive user-friendly interfaces (due to the extensive growth of end-users). Alan Kay wrote:

"... I thought of the whole as the entire computer and wondered why anyone would want to divide it up into weaker things called data structures and procedures. Why not divide it up into little computers, as time sharing was starting to? But not in dozens. Why not thousands of them, each simulating a useful structure?" [51]

The Smalltalk language, developed in the 1970s at Xerox PARC by scientists from the Learning Research Group (LRG), including Alan Kay, became the first formally pure object-oriented programming (OOP) language [38]. Smalltalk was conceived as a dynamic language capable of message exchange between objects. At this time, Xerox PARC developed the concepts of the graphical user interface, desktop computer, monitor, mouse, keyboard, and Ethernet network [52]. In 1973 these concepts materialized in the non-commercial Xerox Alto computer utilizing integrated circuits.

Early methods of object-oriented programming were associated with event-driven techniques, wherein objects were represented as automata controlled by mutual messages to alter their states. In contrast to representing digital entities as structures composed of numbers, these methodologies established a new digital abstraction: a holistic element, an object, that hides the complexity of digital representation from the programmer and instead exhibits its properties for data filling and dynamically interacts with other objects. Despite its lack of popularity in the 1960s, this approach was institutionalized with the advent of personal computers with graphical interfaces in the 1980s and beyond.

In 1967, computer scientist and engineer Ken Thompson introduced a new operating system called UNICS (UNiplexed Information and Computing Service), inspired by his previous work at Bell Labs on developing a multitasking time-sharing operating system for hundreds of users. The UNICS [53] operating system (later abbreviated to UNIX) was initially designed as a modular, structured system oriented towards limited resources, deployment on various systems, and providing access to many users. In 1972, Dennis Ritchie at Bell Labs developed the C programming language [54], which in 1973 became the language of the UNIX kernel. The C language, rooted in ALGOL, gained a clear, minimalist structure and close access to system resources. Later in 1980, the UNIX BSD (Berkeley Software Distribution), implemented the TCP/IP stack developed at Berkeley for the ARPAnet network, contributing to the widespread adoption of TCP/IP. Dennis Ritchie described the communicative aspect of UNIX:

"What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence

of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication." [55]

Later in 1991, Finnish computer science student Linus Torvalds, inspired by MINIX - a small UNIX-like system developed for educational purposes by computer science professor Andrew S. Tanenbaum - introduced Linux [56]. This open-source operating system aimed to provide a free alternative to proprietary UNIX systems and to extend the UNIX approach to a wide range of devices in the future.

At the end of the 1960s, English scientist Edgar F. Codd first described the theory of the relational model of data and relational algebra as an approach to data management [57]. Based on these, in the early 1970s, Donald D. Chamberlin and Raymond F. Boyce at the IBM Research Laboratory in San Jose developed a new declarative data manipulation language called SQL (Structured Query Language), initially named SEQUEL (Structured English Query Language) that for the first time was implemented within IBM's System R database management system [58]. The first customer of System R was Pratt & Whitney in 1977.

Fourth-generation computers integrated computational elements into single-chip microprocessors. In 1971, the Japanese company Busicom, in partnership with Intel, designed and manufactured a chip that first combined all the computational components of a programmable calculator, which previously consisted of several integrated circuits, into a single chip, the Intel 4004 [59], which could execute up to 93,000 instructions per second. In comparison, the ENIAC could execute not more than 5,000 instructions per second. One of the subsequent microprocessors, introduced in 1974, was the legendary Intel 8080, utilized in the first personal computers.

Early microprocessors, including the Motorola 68000 and Intel x86 families, inherited the conceptual Complex Instruction Set Computer (CISC) architecture from mainframes. However, in the mid-1970s and early 1980s, scientists developed the RISC (Reduced Instruction Set Computer) architecture, implemented in the ARM processor family, among others. The RISC architecture provided unified and simpler commands, shifting the complexity of programming internal microcode to external compilers. This allowed a broader range of compiler developers to join the small world of microprocessor developers. The RISC architecture freed up space on the chip for internal memory and number of registers, making RISC processors fast and optimized for specialized and embedded applications, particularly in military, aerospace, medical, and industrial fields. Further competition between CISC and RISC microprocessors stimulated high optimization, support for temporal and concurrent multithreading, and the development of multi-core superscalar microprocessors. Processor specialization spread, leading to processor architectures such as ASIC (Application-Specific Integrated Circuit) and DSP (Digital Signal Processor) aimed at peripheral devices and embedded systems.

In 1974, MITS introduced the first portable computer, the Altair 8800 [60], based on the Intel 8080 processor. The interpreter for it, Altair BASIC, became the first product of Microsoft, founded by Bill Gates and Paul Allen. In 1976, Apple Computer [61], founded by Steve Jobs and Steve Wozniak, constructed its first portable computer, the Apple I. In 1977, the Apple II with an Integer BASIC interpreter was released, becoming the first mass-produced commercial portable computer in the world. In 1981, IBM released the IBM Personal Computer (IBM PC) [62], which became the standard for personal computers for years. The Apple II and IBM PC featured keyboards, mouse manipulators, and video monitors as user interfaces. In 1981, Microsoft introduced its version of a disk operating system, MS-DOS, which became the basic operating system for early personal computers, and in 1985 it introduced the first multitasking graphical environment Windows 1.0 for IBM PC-compatible computers.

In the 1980s, NeXT developed the object-oriented language Objective-C [63] for a new operating system, which served as the foundation for macOS and iOS. Objective-C combined features of the C and Smalltalk languages, particularly supporting event-driven OOP objects designed considering graphical interface tasks. Simultaneously, Bjarne Stroustrup at AT&T Bell Labs began developing the new object-oriented language C++. Unlike Objective-C, C++ first implemented a distinct approach to OOP based on the utilization of methods to define object behavior rather than on event handling [64]. C++ incorporated all modern OOP paradigms: inheritance, polymorphism, and encapsulation. Although C++ did not yet have strict typing, it became the first widely used OOP language.

Monolith Erected from Objects

The field of computer development underwent a profound transformation. The practice of programming began to disseminate from scientific laboratories to a broader audience. Computer science became a popular and accessible subject in universities and colleges, particularly incorporating the paradigms and approaches of object-oriented programming into their curricula. The computer industry transitioned from being primarily developed by government organizations and large corporations to an open market, becoming accessible to home users, education institutions, medium enterprises, applied research laboratories, etc. Novel use cases and application tasks emerged.

One class of applications involved the development of data management and data exchange systems, process management systems, and expert systems for governmental, financial, and banking institutions, manufacturing corporations, research institutions, etc. Thus, in Mannheim, Germany, in 1972, a group of former IBM employees founded SAP (Systems Applications and Products in Data Processing) [65]. In 1978 it combined finance, production, sales, and other resources of large companies in the R/2 computer system. The R/2 systems were acquired by DuPont, General Mills, Goodyear Tire and Rubber, Heinz, Shell, and other giants.

Another class of applications included multitasking windowed operating systems and applications with graphical and other user interfaces for personal computers. In particular, these were office programs, programs for processing and editing audio, graphics, and video, integrated development environments (IDEs), computer-aided design (CAD) systems, programs with graphical interfaces for managing laboratory and medical equipment, computer games, and more. Thus, in 1982, Autodesk entered the market [66] with its inaugural product, AutoCAD for IBM PC.

Embedded applications constitute a distinct class among computer systems. Mobile devices, such as missiles, aircraft, spacecraft, and others, were among the first to require embedded onboard navigation and motion control systems. Likewise, numerical control (NC) applications were needed in manufacturing. Back in 1967, the American company Bedford Associates presented the prototype of a PLC (Programmable logic controller) on semiconductors, the MOdular DIgital CONtroller (Modicon) [67], used by General Motors Hydramatic to control the production of transmissions for Cadillac, Chevrolet, and Pontiac cars. Early embedded systems and applications were experimental military developments or individual industrial and scientific prototypes. The advent of microprocessors led to the widespread propagation of embedded applications in transportation, manufacturing, laboratory research, healthcare, communication. Individual embedded devices eventually combined into automated control systems for various vehicles and large dispatch control systems for transportation, energy, mining, and production facilities. The development of such automation

systems followed predetermined requirements and specified lists of devices, their connections, interfaces, and other characteristics.

Software development processes overall reflected contemporary business organization approaches of the time, namely various forms of end-to-end cascading resource planning for development tasks and teams. Before the development of a program, there were defined stages of requirements analysis and documentation, detailed architectural design, definition of the execution phase, division and coordination of subtasks, followed by sequential phased testing, debugging, and delivery of the final product to the customer or market.

In 1981, the U.S. National Science Foundation (NSF) financially supported the Computer Science Network (CSNET) expansion, which supplemented ARPAnet. In 1982, TCP/IP was standardized as the Internet protocol suite, and network expansion reached a global level. The establishment of the National Science Foundation Network (NSFNet) in 1986 significantly expanded the TCP/IP networks, which in 1988-89 spread to academic and research organizations in Europe, Australia, New Zealand, and Japan. In 1989, Tim Berners-Lee, a British computer scientist and creator of HTTP, HTML, and URI, developed the first version of the HTTP protocol. In 1990, at the European Organization for Nuclear Research (CERN), Berners-Lee launched the first web server, CERN httpd. The World Wide Web emerged [68, 69].

The proliferation of networks, and eventually the global Internet, established the most popular architectural model of that time: the "client-server" model. The genesis of the "client-server" architecture can be traced back to the separation of the first user interfaces from the computing core and the appearance of remote terminals. Network systems laid down formal principles for interaction between clients and servers. Although the efficiency of dividing the system into central "thick" and peripheral "thin" processes favored the client-server organization of processes even within a single computer, the development of networks led to a model in which there is a central computer (server) in the network and other computers (clients) that send requests to the server and receive responses available to the user in desktop or browser applications. The emergence of HTTP servers further expanded the range of use cases and application tasks. A transition commenced from client desktop applications previously available on local networks to HTTP clients available on global networks. Thus, web servers implemented tasks of data and content management, resource management, and email, and made early attempts to reach customers through e-commerce stores.

In 1991, Sun Microsystems initiated the development of a novel OOP language, Java, which inherited the syntax of C and C++. Anticipating the global spread of networks, Java was designed to be a platform-independent programming language for any kind of device, from server computers to chips embedded in newly appearing mobile phone cards. The concept of the Java Virtual Machine (JVM) [70, 71] was formally defined, comprehensively isolating the hardware and operating resources of the computer from the Java programmer. For effective operation of the JVM, Java implemented completely static OOP, expecting explicit type definitions for the successful compilation of a program. Java collections designed upon optimal sorting and search algorithms were developed to support multiple objects. The language became straightforward to learn, with the implementation of complex algorithms encapsulated and hidden from the programmer, who needed to use ready-made types or their inheritance. The Java OOP approach did not gain much traction in GUI applications. Instead, it became particularly popular in the area of server solutions. At the end of the 1990s, Microsoft introduced the C# language [72], similar in approach to Java.

Server solutions did not possess graphical interfaces and did not implement active object interactions. The programming of objects involved reflecting a business task into separate

statically related classes, such as "warehouse," "product," "customer," "invoice," etc., for which necessary data and processes were defined. Datasets were formed by SQL queries to a relational database. Requests from the client application served as entry points to specific server processes. Upon receipt of a request, the server processed and composed data, generated elements of the visual representation of the response, and provided it back to the client according to the request parameters. The digital representation formed the primary data and system behavior in basic abstract classes, while a variety of functionalities supported practical scenarios of handling numerous requests in complex class hierarchies and their projections onto interrelated database table structures. It proved impossible to separate such scenarios from each other. Therefore, the system was necessarily constructed as a monolithic structure.

As was the case with procedural programming, a situation arose where the implementation of new types of applications utilizing OOP languages became increasingly challenging. The inheritance and encapsulated functionality reuse paradigm, which originally simplified code structure, now stimulated the creation of complex class hierarchies, leading to significant code coupling and, consequently, difficulty in making any changes to the program. The principal methods for organizing the code of complex applications became programming patterns and various rules for writing good OOP code [73], such as KISS (Keep It Simple, Stupid), DRY (Don't Repeat Yourself), YAGNI (You Aren't Gonna Need It), BDUF (Big Design Up Front), APO (Avoid Premature Optimization), SOLID, and others, all based on the generalization of practical experience in applying OOP by skilled developers rather than some formal evaluations.

Another drawback of monolithic server solutions was their performance limitations. As the number and complexity of processes for handling client requests increased, high latency and overloading of typical server performance became common issues. To significantly improve overall system performance, server hardware performance had to be increased.

Remaining an Object Among Virtual Machines

At the beginning of the 2000s, client-server solutions became widely popular among applications for medium and small businesses, supermarkets, schools, hospitals, social institutions, local governments, etc., primarily for resource management, accounting, and supporting operational and commercial activities. These organizations used inexpensive, commercially available hardware systems. However, by the early 2000s, these existing single-processor servers generally could not deliver the performance gains needed for monolithic applications. For these reasons, and due to the growing globalization of business and the merging of local enterprises into large business structures, client-server applications began to specialize. Different segments and departments of enterprises and organizations deployed their specialized server applications on separate servers, representing interfaces for interaction known as web services. To enable the interaction of such services and combine them into a single system a new class of specialized solutions emerged - enterprise service bus (ESB). A service-oriented architecture (SOA) [74] was established, which generally supported the utilization of monolithic applications as particular services while enabling the deployment of the entire system on numerous individual server computers.

The early computers employing a set of parallel processors for computations within a shared memory were introduced in the 1960s-1970s. Notable examples include the supercomputers developed by American engineer Seymour Cray [75]. However, these were costly, custom-built products purposefully designed for intensive scientific calculations. With the improvement of microprocessors in the 1980s and 1990s, less expensive business-class solutions appeared, including the IBM POWER microprocessor architecture family. In the

2000s, these innovations led to the development of parallel and distributed computing systems utilizing multi-core microprocessors and combining multiple processors and individual servers into powerful computing architectures. Subsequently, such computing systems have specialized in two distinct directions: virtual computing environments for a wide range of business applications, and specialized parallel computing systems - modern supercomputers - for use in scientific research centers, etc.

In 1999, VMware introduced a software application that enabled the execution of multiple instances of x86 or x86-compatible operating systems to run on a single physical computer [76]. In 2002, Amazon established the AWS platform, which, since 2006, has provided commercial access to remote data centers with data storage systems and virtual machines for deploying applications. In 2008, Microsoft established its remote deployment platform, Windows Azure, and Google introduced the App Engine. The concept of "cloud computing" emerged.

In 2000, American scientist Roy Thomas Fielding, one of the authors of HTTP and URI specifications, presented his doctoral dissertation entitled "Architectural Styles and the Design of Network-based Software Architectures" at the University of California [77]. In chapter five, "Representational State Transfer (REST)," he suggested a conceptual framework over HTTP that defined the atomicity of entities on the network and established rules for changing their state, laying the foundation for the RESTful concept of application-level protocols.

In 2004, engineer Eric Evans, in his book "Domain-Driven Design: Tackling Complexity in the Heart of Software," [78] described principles for structuring any application task's domain (problem) area to be represented in a digital solution. He proposed a method for constructing a set of loosely coupled, meaningfully homogeneous bounded contexts and developing an abstract service structure that represents such a context and provides an interface for modifying its state.

In the mid-2000s, a novel approach to designing distributed server systems emerged, distinct from both monolithic and service-oriented architectures. The microservice [79, 80, 81] became the basis of this new approach, encompassing an identified domain entity and representing a synchronous RESTful interface and/or an asynchronous messaging interface for modifying its state. The data representing a domain entity do not possess the complex internal links typical of relational database representations. Instead, they can be represented, for example, as a document or key-value pair, as is inherent in non-relational data management systems. The microservice reflects a meta-object representation of a domain entity, where the “meta-objects” dynamically process messages or change states among themselves or in response to user requests. Essential business logic forming visual representations can be extracted from the server to the client application, for example, using technologies such as Single Page Applications (SPA). In a sense, the microservices resemble dynamic objects implemented in the SmallTalk language yet but operate at a new level of abstraction.

To support the deployment of microservice systems, environment preparation (terraforming) and containerization technologies were developed alongside virtualization. Cloud platforms introduced access and administration of cloud infrastructure and enabled pre-deployed applications for data and message management, system auto-scaling, etc., establishing the infrastructure, platforms, and software as a service (IaaS, PaaS, SaaS) approach. A new direction in programming systems in remote environments, known as development and operations (DevOps), emerged. The term "DevOps" first gained traction in 2009 during the DevOps Days meetings in Belgium [82].

OOP languages such as Java, C#, and C++ continued to be widely utilized for writing microservices but now without the need for building intricate class hierarchies. In the absence of a need for formalized OOP, languages supporting multiple programming paradigms

simultaneously (procedural, object-oriented, aspect-oriented, and functional) are effectively developing today. Among new and experimental languages, these include Python, Perl, Ruby, Scala, R, and others. This mixed paradigm suggests utilizing a single language environment to write and deploy microservices and create logic for business and scientific tasks [83-89].

In 2001, leaders of prominent software development methodologies endorsed the Agile Manifesto. The agile development principles replaced end-to-end cascade development with sequential approaches and continuous adaptation of project requirements based on already gained project experience and achievements [90, 91]. The further development of agile methodologies is moving towards self-organization.

Due to the explosive growth of remote computing systems in the 2010s and 2020s, alongside the widespread digitalization of various aspects of human life and activity, the application tasks of computing systems expanded to global business applications and applications for large data and computational tasks in applied and fundamental scientific research. Such systems are employed in the collection, storage, and analysis of large data sets in a variety of fields, including genome research, molecular modeling, elementary particle collision analysis, exoplanet detection, imaging black holes, machine vision, virtual and augmented reality, supporting blockchain, and social networks. Due to the powerful capabilities of parallel and distributed computing, artificial intelligence research and applications are experiencing an intensive revival, including renewed interest in large language models (LLM) and generative AI applications [92].

Everyone Needs a Digital Twin

Alongside the development of remote data centers and supercomputers, a new phenomenon of digital representation emerged on the verge of the third millennium. This was enabled by the sustained expansion of global networks, the miniaturization of computers, and the opening of a new realm of user applications in global digitalization.

The applications of embedded solutions and automation systems, previously utilized in scientific, military, industrial, and medical fields, began to extend to consumer electronics and communication at the turn of the 1990s. For example, in 1983, Motorola introduced the prototype mobile phone Dynatac 8000x [93], and as early as 1987, the GSM digital cellular communication standard was adopted [94]. In 1984, Sony released the first portable compact disc player, the Discman D-50/D-5 [95]. In 1988, Fuji and Toshiba introduced the first full-fledged digital camera, the Fuji DS-1P [96]. In the 1990s, digital controllers became ubiquitous in household devices such as washing machines, kitchen appliances, climate control systems, entertainment systems, etc.

With the early advent of networks, consumer appliances could be connected to them. In 1982, a modified Coca-Cola vending machine was connected to the ARPAnet network to share information about drinks inventory and temperature. However, unlike traditional automation, the integration of consumer and household devices into automated systems required the flexibility to modify and expand the system during use, the integration of standards from different manufacturers, and the availability of universal control interfaces that are understandable to home users. Specifying all the components and connections at the design stage enabled the creation of static individual solutions that were in limited demand.

The term "Internet of Things" (IoT) was first introduced in a September 1985 speech at the Congressional Black Caucus Foundation's 15th Annual Legislative Weekend by Peter T. Lewis [97], who defined IoT as the integration of people, processes, and technology with connected devices and sensors, enabling remote monitoring, status checking, manipulation, and trend

analysis of such devices. British engineer Kevin Ashton independently introduced the term "Internet for Things" in 1999 [98].

In the subsequent decades, protocols, secure connection interfaces, system integration methods, and ways of digitally representing things were developed. The field of smart things and smart solutions integrating smart things into holistic systems has emerged. One such approach to developing smart solutions was to connect smart things to remote cloud services and create a "digital twin" of the devices involved, allowing their configuration and current state to be viewed and modified. In 2015, the AWS cloud platform launched AWS IoT services, and in 2016, the Microsoft Azure platform launched Azure IoT Hub services, both supporting the concept of digital twins of devices.

The domain of smart things and smart solutions has become a primary driver of global digitalization nowadays. It is therefore necessary to develop novel methodologies for designing and developing systems that align with the openness and dynamism inherent in the use cases and application tasks of the smart things and smart solutions domain. So, traditional automation systems already implemented in industrial enterprises demonstrate a deficiency of flexibility in applying new solutions involving multiple integrations of autonomous devices, installations, plants, big data processing, smart control scenarios, and predictive maintenance using artificial intelligence, among other novel use cases. The issue of transitioning between different stages of developing industrial systems is typically addressed in the meta-standards of industrial automation, such as Industry 3.0 and Industry 4.0 [99].

The most contemporary applications of autonomous device systems generally extend beyond the scope of mere networks of things. A representative example of these new tasks is the recently submitted policy brief by the European Union, "Industry 5.0: A Transformative Vision for Europe" (2022) [100], which outlines the objectives for the next generation of the global European industrial-economic approach, which, in contrast to the previous machine-centric approach, now aims at a human-centric production organization that considers actual consumption, resilience to new shocks (such as pandemics, natural disasters, geopolitical changes, regional wars, etc.), focus on environmental concerns, and the ability to self-organize. Designing such solutions entails interpreting "digital twins" of systems and phenomena not merely as "twins" of purely technical devices, but as complex entities of systemic, societal, environmental, economic, and other nature. These entities receive data from various sources, including energy, production, environmental sensors, and products during their entire lifecycle.

Nowadays and soon, the tasks of complex distributed systems include solutions for smart residences and cities; smart society, government, and economics; smart healthcare and environmental systems; smart distributed and renewable energy systems; smart space and celestial body exploration systems; smart drone systems (swarms); smart systems of ultra-miniature devices (nanobots); and others.

A review and analysis of practical architectural development experience in this field reveals the following major trends in the development of such future systems:

- The number of autonomous elements tends to be large.
- The diversity of autonomous elements tends to match their total number.
- When they are numerous, autonomous elements tend to evenly distribute complexity.

This suggests that such systems generally may not depend on centralized data centers or networks but instead distribute decision-making, computing, and data storage among their autonomous elements.

The extant methods for representing digital entities, such as functional, procedural, and object-oriented programming, as well as domain-driven design, demonstrate inherent constraints

in modeling complex distributed systems. For instance, the static OOP approach necessitates the pre-definition of all entity types at the compilation stage. The diverse forms of OOP and DDD permit the only qualitative definition of classes or bounded contexts using descriptive categories, typically at the discretion and experience of the engineer. The structural characteristics of digital representations in OOP are limited to planar object properties. Collections of objects are designed for deployment in a single computer environment, and remote calls can only ensure a transfer of the state of such collection elements across the network.

Thus, the current state of technology does not support the formation of a comprehensive solution that comprises an open set of individual autonomous devices. Additionally, existing methodologies do not provide the ability to accessibly represent and model such a system with all the characteristics as a single whole.

A possible solution to this challenge is an approach to the representation of a system based on the principles of the post-non-classical paradigm. This approach anticipates a shift from the necessity of initial particularity of types, structures, and components, but postulates the original individual exclusivity, distinctiveness, and unstructured nature of elements of a system.

Here, a primary aspect of the representation of a system is a potential for unbounded variation and interaction of elements, while the particular definition of structures and types of elements is a secondary aspect which is aimed at the parameterization of a system according to a certain purpose. Such interventions of particularity can be, for example, the interactions that are aimed at observing or modifying a system. One source of the interaction is a user, who becomes a participant in the digital representation environment. Therefore, representing interaction through formal (mathematical) dependencies, operations, and transformation rules will characterize the system qualitatively and quantitatively.

The fundamental abstraction of representation in a digital environment should be defined as the Digital Entity Origin (DEO).

For DEO, the following postulates should be defined:

Boundless and Limitless Entity. DEO comprises specific aspects of an entity, namely attributes associated in an organization that have no predefined boundaries or structure. A mathematical representation of DEO can be expressed as an attributed graph, where edges represent systems of attribute associations of DEO, and vertices represent the state of each attribute as a nature of interaction potential according to formal (mathematical) systems of operations (state as an interaction). The whole system of attributes of DEO conceptually reflects the volume of DEO.

Only instances matter. DEO represents an entity that exists (real or virtual), rendering each DEO particular and not determining its type (or class, etc.) by definition. To achieve individual DEOs into a uniform representation within some set (type), it is necessary to conform (fold/unfold) the volume of each DEO according to formal (mathematical) systems of operations to a common volume in the desired set (type as an interaction).

Expansion, not extension. DEOs' alteration is achieved through their transformations (such as divisions, unions, etc.) into new DEOs with their respective volumes without reusing or hiding the volumes of other DEOs (as inheritance or encapsulation). The formation or transformation of DEOs occurs according to formal (mathematical) systems of operations (definition as an interaction). The propagation of DEO creates compositions of many DEOs, eventually forming the DEO space.

To illustrate a targeted system with the ability to self-control, self-organize, and self-improve, one might consider a mining station situated in the asteroid belt. In this scenario, various types of drones and their constituent parts, as well as asteroids and their fragments, are

represented by compositions of DEO entities. The formalized interaction of all DEO attributes enables the system to continuously maintain positioning in space and distances between objects as well as perform self-organized technological operations such as surface scanning and mineral analysis, mining and enrichment, accumulation, and logistics. The system dynamically positions DEO representations on individual devices; for example, the DEO of a specific rock fragment is sequentially located in different mining devices. The system distributes computations among individual devices and dynamically adapts communication. In the event of system disturbances, such as the physical loss or acquisition of devices, DEO space transformations occur, and reorganized interactions allow the mission to be completed.

Further research is needed to develop a comprehensive specification and formal description of a digital entity representation system, such as DEO space. Additionally, the creation of practical design tools within the DEO paradigm and the subsequent adaptation of technologies for implementing such systems are necessary.

Conclusion

The representation of domain entities in computing environments has evolved alongside technological advancements and the emergence of new application tasks. In the early stages, computations consisted of operations and operands. Simple operations evolved into command sequences and acquired complex organization through procedural and structured programming. Simple operands evolved into diverse data structures, ranging from linear arrays to relational and graph databases. Eventually, there was a transition to representing "small computers" within a computer - objects. Object-oriented programming evolved from message exchange in graphical interfaces and network system simulations to complex static hierarchical class structures in monolithic server applications. With the advent of remote computing, the domain-driven representation of bounded contexts for correspondent use cases was established. Domain-driven design and microservices technology are contemporary methods for representing systems in cloud applications.

The representation of digital entities in distributed systems, which consist of autonomous devices with embedded computers and network connectivity, marks a distinct direction in information technology and computer science. One of the approaches to representing such systems is to design "digital twins" of these devices. As the complexity of application tasks for distributed systems grows, the increasing number and diversity of devices, abstraction of entities, self-control, self-organization, self-improvement, decentralization, and both quantitative and qualitative characteristics of these systems necessitate a novel holistic approach to their description and representation.

References

[1] *T. Collins*, The Legendary Model-T Ford: The Ultimate History of America's First Great Automobile, *Krause Publications*, 2024

[2] *J. Paxton*, “Mr. Taylor, Mr. Ford, and the Advent of High-Volume Mass Production: 1900-1912,” *Economics & Business Journal*: *vol.4 no.1* Oct 2012

[3] *C. Akmut*, From selling to ‘thinking’ the Machine: Pascal's intellectual trajectory, sociological aspects, osf preprints, May 1, 2020, doi:10.31219/osf.io/qfc5v

[4] *Y. Serra*, La machine arithmétique de Leibniz, *OpenEdition Journals: Bibnum Textes fondateurs de la science Calcul et informatique*, 2017. doi:10.4000/bibnum.551

[5] *N. K. Tailor*, "Charles Babbage's Mini-Computer - Difference Engine," *No. 0 IMA Bulletin, 28 (6)*, pp. 112-114, 1992

[6] *A. G. Bromley*, "Charles Babbage's Analytical Engine, 1838," *Annals of the History of Computing, vol. 4, no. 3*, pp. 196-217, Jul-Sep 1982. doi: 10.1109/MAHC.1982.10028

[7] *R. Rojas*, "Konrad Zuse's legacy: the architecture of the Z1 and Z3," *IEEE Annals of the History of Computing, vol. 19, no. 2*, pp. 5-16, Apr-Jun 1997. doi: 10.1109/85.586067

[8] *P. E. Ceruzzi*, The Early Computers of Konrad Zuse, 1935 to 1945, *Annals of the History of Computing, vol. 3, no. 3*, pp. 241-262, July-Sep 1981. doi: 10.1109/MAHC.1981.10034

[9] *H. Aiken, A. G. Oettinger, T. C. Bartee*, Proposed automatic calculating machine, *EEE Spectrum, vol. 1, no. 8*, pp. 62-69, Aug 1964. 10.1109/MSPEC.1964.6500770

[10] *T. Sale*, Lecture given at the IEEE 18th [Electronic Resource] Feb 1999, - [Cited 2024, 22 May]. - Available from: https://www.codesandciphers.org.uk/lectures/ieee.txt

[11] *B. Copeland*, The Manchester Computer: A Revised History Part 1: The Memory, *IEEE Annals of the History of Computing, vol. 33, no. 1*, pp. 4-21, Jan 2011 doi: 10.1109/MAHC.2010.1

[12] *B. Copeland*, The Manchester Computer: A Revised History Part 2: The Baby Computer, *IEEE Annals of the History of Computing, vol. 33, no. 1*, pp. 22-37, Jan 2011, doi: 10.1109/MAHC.2010.2

[13] 10. The Ferranti Computer at Manchester University, England, *Digital Computer Newsletter. 3 (3): 4–5.* Oct 1951

[14] 11. The Ferranti Computer at Manchester University, England, *Digital Computer Newsletter. 4 (3): 6.* Jul 1952

[15] *J. W. Cortada*, The ENIAC's influence on business computing, 1940s-1950s, *IEEE Annals of the History of Computing, vol. 28, no. 2*, pp. 26-28, Apr-Jun 2006. doi:10.1109/MAHC.2006.38

[16] *M. Wilkes*, What I remember of the ENIAC, *IEEE Annals of the History of Computing, vol. 28, no. 2*, pp. 30-31, Apr-Jun 2006. doi: 10.1109/MAHC.2006.41

[17] G. O'Regan, EDVAC and ENIAC Computers, *The Innovation in Computing Companion. Springer 09*, pp 113–117, Dec 2018. doi:10.1007/978-3-030-02619-6_23

[18] *L. P. Tabor*, The EDVAC, an electronic digital computer, *Astronomical Journal, Vol. 53*, p. 205, Jan 1948. doi:10.1086/106125

[19] *D. E. Lundstrom*, A Few Good Men From Univac, *The MIT Press*, 264 pp, 1987

[20] *R. R. Everett*, The Whirlwind I computer, *Papers and discussions presented at the Dec. 10-12, 1951, joint AIEE-IRE computer conference: Review of electronic digital computers (AIEE-IRE '51). Association for Computing Machinery, New York, NY, USA*, 70–74. doi:10.1145/1434770.1434781

[21] *C. E. Frizzell*, Engineering Description of the IBM Type 701 Computer, *Proceedings of the IRE, vol. 41, no. 10*, pp. 1275-1287, Oct 1953. doi:10.1109/JRPROC.1953.274301

[22] *C. C. Hurd*, Early Computers at IBM, *Annals of the History of Computing, vol. 3, no. 2*, pp. 163-182, Apr-Jun 1981. doi:10.1109/MAHC.1981.10021

[23] IBM 701 Customers, *IBM Archives.* [Electronic Resource] 23 Jan 2003, - [Cited 2024, 14 Feb]. - Available from: https://www.ibm.com/ibm/history/exhibits/701/701_customers.html

[24] *S. H. Kaisler*, First Generation Mainframes: The IBM 700 Series, *Cambridge Scholars Publishing*, 27 Feb 2018

[25] *P. McJones*, History of FORTRAN and FORTRAN II, *Software Preservation Group* [Electronic Resource] - [Cited 2024, 20 May]. - Available from: https://www.softwarepreservation.org/projects/FORTRAN/

[26] *S. L. Andresen*, "John McCarthy: father of AI Intelligent Systems," *IEEE. 17. 84 - 85*, 2002. doi:10.1109/MIS.2002.1039837

[27] Europe's first computer, *NCUBE* [Electronic Resource] - [Cited 2024, 22 May]. - Available from: https://ncube.com/tag/mesm

[28] *M. Hally*, Electronic Brains. Stories from the Dawn of the Computer Age, *Joseph Henry Press*, Sep 23, 2005

[29] *H. H. Goldstine*, The Computer from Pascal to von Neumann, *Princentom Univercity Press, Oct 21*, 1980

[30] *R. Rojas*; U. Hashagen,the First Computers. History and Architectures", *The MIT Press*, July 26, 2002

[31] *M. Riordan, L. Hoddeson*, Crystal fire: the invention, development and impact of the transistor, *IEEE Solid-State Circuits Society Newsletter, vol. 12, no. 2*, pp. 24-29, Spring 2007. doi:10.1109/N-SSC.2007.4785574

[32] *E. W. Pugh, L. R. Johnson, John H. Palmer*, IBM's 360 and Early 370 Systems, *The MIT Press, Jan 1, 2003*

[33] *Weik, Martin H.*, A Fourth Survey of Domestic Electronic Digital Computing Systems. Ballistic Research Laboratories, Aberdeen Proving Ground, Jan 1964, [Electronic Resource] - [Cited 2024, 11 Mar]. - Available from: https://www.ed-thelen.org/comp-hist/BRL64-m.html#MINUTEMAN

[34] M. Mattioli, The Apollo Guidance Computer *IEEE Micro, vol. 41, no. 6*, pp. 179-182, 1 Nov-Dec 2021. doi:10.1109/MM.2021.3121103

[35] *M. Kahn*, The Beginning of I.T. Civilization – IBM's System/360 Mainframe, *Navigating Information Technology Horizons*, 2001, [Electronic Resource] - [Cited 2023, 2 Aug]. - Available from: http://www-07.ibm.com/systems/tw/z/download/clipper_mainframe_at40.pdf

[36] *W. H. Blair*, The 360/91 and associated machines, *Narkive Mailing List Archive*, 2008, [Electronic Resource] - [Cited 2024, 5 Mar]. - Available from: https://hercules-390.yahoogroups.narkive.com/ritmdhO6/the-360-91-and-associated-machines

[37] *P. McJones*, History of ALGOL, *Software Preservation Group* [Electronic Resource] - [Cited 2024, 21 May]. - Available from: https://www.softwarepreservation.org/projects/ALGOL/

[38] *P. McJones*, History of LISP, *Software Preservation Group* [Electronic Resource] - [Cited 2024, 19 May]. - Available from: https://www.softwarepreservation.org/projects/LISP/

[39] *J. E. Sammet*, The real creators of Cobol, *IEEE Software, vol. 17, no. 2*, pp. 30-32, Mar-Apr 2000. doi: 10.1109/52.841602

[40] MVS JCL Reference, *IBM Corporation*, 2022 [Electronic Resource] - [Cited 2024, 5 Mar]. - Available from:https://www.ibm.com/docs/en/zos/2.2.0?topic=mvs-zos-jcl-reference

[41] *D. Walden, T. Van Vleck*, The Compatible Time Sharing System (1961–1973). Fiftieth Anniversary. Commemorative Overview", *IEEE Computer Society*, 2011

[42] *J. McCarthy, S. Boilen, E. Fredkin, J. C. R. Licklider*, A time-sharing debugging system for a small computer, *Proceedings of the May 21-23, 1963, spring joint computer conference (AFIPS '63 (Spring)). Association for Computing Machinery, New York, NY, USA, 51–57*, 1963. doi:10.1145/1461551.1461559

[43] J. *G. Kemeny, T. E. Kurtz*, Dartmouth Time-Sharing, *Science 162,223-228*, 1968. doi:10.1126/science.162.3850.223

[44] *T. E. Kurtz*, BASIC. History of programming languages, *Association for Computing Machinery, New York, NY, USA*, 515–537, 1987. doi:10.1145/800025.1198404

[45] *C. Paloque-Bergès, V. Schafer*, Arpanet (1969–2019), *Internet Histories, 3(1), 1–14*, 2019 . doi:10.1080/24701475.2018.1560921

[46] *G. Strawn*, Masterminds of the Arpanet, *IT Professional, vol. 16, no. 3*, pp. 66-68, May-Jun 2014. doi:10.1109/MITP.2014.32

[47] *I. E. Sutherland*, "Sketchpad, a man-machine graphical communication system. Thesis (Ph. D.)", *Massachusetts Institute of Technology*, 1963

[48] *K. Nygaard, O.-J. Dahl*, The Development of the SIMULA Languages, *ACM SIGPLAN Notices. Vol. 13. No. 8.* Aug 1978

[49] *K. Nygaard*, SIMULA--an extension of ALGOL to the description of discrete event networks, *Munich paper*, 1963, p. 520 [Electronic Resource] - [Cited 2024, 20 May]. - Available from: https://users.dcc.uchile.cl/~cgutierr/cursos/LP/SimulaHistory.html

[50] *G. E. Moore*, Cramming more components onto integrated circuits, *Reprinted from Electronics, vol. 38, no.8*, Apr 19, 1965, pp.114 ff, *IEEE Solid-State Circuits Society Newsletter, vol. 11, no. 3, pp. 33-35*, Sep 2006. doi:10.1109/N-SSC.2006.4785860

[51] *A. C. Kay*, The Early History Of Smalltalk, *Association for Computing Machinery SIGPLAN Notices, Volume 28, No. 3,* Mar 1993, p.71 (printed p.6). doi:/10.1145/155360.155364

[52] *M. A. Hiltzik*, Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age, *Harper Business; First Edition*, Apr 5, 2000

[53] *P. Gethin*, UNIX — Where it came from and where it is going, *VINE, Vol. 20 No. 3*, pp. 4-7, 1990. doi:10.1108/eb040438

[54] *D. M. Ritchie*, The development of the C language, *The second ACM SIGPLAN conference on History of programming languages (HOPL-II). Association for Computing Machinery, New York, NY, USA*, 201–208, 1993. doi:/10.1145/154766.155580

[55] *D. M. Ritchie*, The UNIX system: The evolution of the UNIX time-sharing system, *AT&T Bell Laboratories Technical Journal, vol. 63, no. 8,* pp. 1577-1593, p.1578, Oct 1984. doi:10.1002/j.1538-7305.1984.tb00054.x.

[56] *H.J. Meeker*, Introduction: How Unix Gave Birth To Linux, And A New Software Paradigm. In The Open Source Alternative, *John Wiley & Sons*, 2012. doi:10.1002/9781119197706.ch1

[57] *E. F. Codd*, A relational model of data for large shared data banks *Commun. ACM 13, 6,* 377–387, Jun 1970. doi:10.1145/362384.362685

[58] *D. D. Chamberlin and others*, A history and evaluation of System R, *Commun. ACM 24, 10*, 632–646. Oct 1981. doi:10.1145/358769.358784

[59] The Intel 4004 Microprocessor and the Silicon Gate Technology, [Electronic Resource] - [Cited 2024, 6 Mar]. - Available from: https://users.dcc.uchile.cl/~cgutierr/cursos/LP/SimulaHistory.htmlhttp://www.intel4004.com/proto_calc.htm

[60] *H. Garland*, Design Innovations in Personal Computers, *Computer, vol. 10, no. 3,* pp. 24-27, March 1977. doi:10.1109/C-M.1977.217669

[61] *S. Wozniak*, IWoz: computer geek to cult icon : how I invented the personal computer, co-founded Apple, and had fun doing it, *New York : W.W. Norton & Co*, 2006

[62] *G. Singh*, The IBM PC: The Silicon Story, *Computer, vol. 44, no. 8,* pp. 40-45, Aug 2011. doi:10.1109/MC.2011.194

[63] B. *J. Cox, S. Naroff, Hansen Hsu*, The origins of Objective-C at PPI/Stepstone and its evolution at NeXT, *Proc. ACM Program. Lang. 4, HOPL, Article 82*, Jun 2020, 74 pages. doi:10.1145/3386332

[64] *B. Stroustrup*, A history of C++: 1979–1991, *SIGPLAN Not. 28, 3*, Mar1993, 271–297. doi:10.1145/154766.155375

[65] *T. Leimbach*, The SAP Story: Evolution of SAP within the German Software Industry, *IEEE Annals of the History of Computing, vol. 30, no. 4,* pp. 60-76, Oct-Dec 2008. doi:10.1109/MAHC.2008.75

[66] *D. Weisberg*, History of CAD, [Electronic Resource] - [Cited 2024, 18 Mar]. - Available from: https://www.shapr3d.com/blog/history-of-cad?utm_campaign=cadhistorynet

[67] *A. Dunn*, The father of invention: Dick Morley looks back on the 40th anniversary of the PLC, *Manufacturing Automation,* Jun 12, 2009 [Electronic Resource] - [Cited 2024, 22 May]. - Available from: https://www.automationmag.com/855-the-father-of-invention-dick-morley-looks-back-on-the-40th-anniversary-of-the-plc/

[68] *E. Rabinovitch*, The language Of The Internet, *IEEE Communications Magazine, vol. 36, no. 2,* pp. 24-26, Feb. 1998. doi:10.1109/MCOM.1998.648750

[69] *T. Berners-Lee et al*, Creating a Science of the Web, *Science Vol 313, Issue 5788* pp. 769-771, 2006. doi: 10.1126/science.1126902

[70] *J. Meyer*, Java Virtual Machine 1st Edition, *O'Reilly Media; 1st edition* Apr 11, 1997

[71] *J. R. Hines*, Virtual machines jockey for position, *IEEE Spectrum, vol. 34, no. 7*, pp. 16-16, Jul 1997. doi:10.1109/MSPEC.1997.609806

[72] *D. R. Naugler*, C# 2.0 for C++ and Java programmer: conference workshop, J. *Comput. Sci. Coll. 22, 5, 1.*, May 2007

[73] *N. K. Nghiem*, SOLID, KISS, YAGNI and DRY Principles [Electronic Resource] - [Cited 2023, 13 Nov]. - Available from: https://dev.to/nknghiem/solid-kiss-yagni-and-dry-principles-ie7

[74] *K. Kontogiannis*, G. A. Lewis, D. B. Smith, A research agenda for service-oriented architecture, *Proceedings of the 2nd international workshop on Systems development in SOA environments (SDSOA '08). Association for Computing Machinery, New York, NY, USA, 1–6.* doi:10.1145/1370916.1370917

[75] *G. Strawn & C. Strawn*, The Father of Supercomputing: Seymour Cray, *T Professional, vol. 17, no. 2*, pp. 58-60, Mar-Apr 2015. doi:10.1109/MITP.2015.31

[76] *E. Bugnion et al*, Bringing Virtualization to the x86 Architecture with the Original VMware Workstation, *ACM Trans. Comput. Syst. 30, 4, Article 12*, 51 pages, Nov 2012. doi:10.1145/2382553.2382554

[77] *R. T. Fielding & R. N. Taylor*, Architectural styles and the design of network-based software architectures. Ph.D. Dissertation, *University of California, Irvine*, 2000. doi:10.5555/932295

[78] *E. Evans*, Domain-Driven Design: Tackling Complexity in the Heart of Software, *Addison-Wesley Professional; 1st edition* Aug 20, 2003

[79] *P. Jamshidi et al*, Microservices: The Journey So Far and Challenges Ahead, *IEEE Software, vol. 35, no. 3*, pp. 24-35, May/Jun 2018. doi:10.1109/MS.2018.2141039

[80] *D. Shadija, M. Rezai & R. Hill*, Towards an understanding of microservices, *23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK*, 2017, pp. 1-6. doi:10.23919/IConAC.2017.8082018

[81] *A. Sill*, The Design and Architecture of Microservices, *IEEE Cloud Computing, vol. 3, no. 5*, pp. 76-80, Sep-Oct 2016. doi:10.1109/MCC.2016.111

[82] About devopsdays, [Electronic Resource] - [Cited 2024, 20 May]. - Available from: ttps://devopsdays.org/about/

[83] *M. Goliński & A. K. Golińska*, Ruby vs. Perl – the Languages of Bioinformatics, *Studies in Logic, Grammar and Rhetoric 35 (1):*143-155, 2013. doi:10.2478/slgr-2013-0032

[84] *Yukihiro Matsumoto*, From Lisp to Ruby to Rubinius, *Workshop on Self-Sustaining Systems (S3 '10). Association for Computing Machinery, New York, NY, USA, 9.* doi:10.1145/1942793.1942795

[85] *T. Jenkins*, The First Language - A Case for Python?, *Innovation in Teaching and Learning in Information and Computer Sciences, 3(2), 1–9.*, 2004. doi:10.11120/ital.2004.03020004

[86] *P. Kwan*, Why Scala matters: introduction to Scala programming, *J. Comput. Sci. Coll. 31, 1*, 108–109, Oct 2015. doi:10.5555/2831373.2831391

[87] *J. A. Miller, J. Han, M. Hybinette*, Using Domain Specific Language for modeling and simulation: ScalaTion as a case study, *Proceedings of the 2010 Winter Simulation Conference, Baltimore, MD*, USA, 2010, pp. 741-752. doi:10.1109/WSC.2010.5679113

[88] *FM. Giorgi, C. Ceraolo, Mercatelli D*, The R Language: An Engine for Bioinformatics and Data Science, Life. *12(5):648,* 2022. doi:10.3390/life12050648

[89] *T. L. Staples*, Expansion and evolution of the R programming language, *Royal Society Open Science. V.10, I.4, 12* Apr 2023. doi:10.1098/rsos.221550

[90] *A. K. Kakar*, A Rhetorical Analysis of the Agile Manifesto on its 20th Anniversary, *The Journal of the Southern Association for Information Systems, 10, 20-29.* doi:10.17705/3JSIS.00030

[91] *D. Øivind Madsen*, The Evolutionary Trajectory of the Agile Concept Viewed from a Management Fashion Perspective, *Social Science, 9(5), 69,* 2020. doi:10.3390/socsci9050069

[92] M. A. Haque, A Brief Analysis of 'ChatGPT' – A Revolutionary Tool Designed by OpenAI, *EAI Endorsed Trans AI Robotics, vol. 1, p. e15,* Mar. 2023. doi:10.4108/airo.v1i1.2983

[93] *S. Miller*, Motorola Executive Helped Spur Cellphone Revolution, Oversaw Ill-Fated Iridium Project, *The Wall Street Journal*, Jun 20, 2009 [Electronic Resource] - [Cited 2024, 20 May]. - Available from: https://www.wsj.com/articles/SB124546835819133721

[94] *A. A. Huurdeman*, The Worldwide History of Telecommunications, *Wiley-IEEE Press; 1st edition*, Jul 31, 2003, p. 529. doi:10.1002/0471722243

[95] Sony Portable CD-i Player - The intelligent discman, [Electronic Resource] - [Cited 2024, 23 May]. - Available from: https://smallmart.nl/artikelen/vintage-computers/68-sony-portable-cd-i-player-the-intelligent-discman

[96] A photographic film producer develops the world's first fully digital camera [Electronic Resource] - [Cited 2024, 23 May]. - Available from: http://www.fujifilm.com/innovation/achievements/ds-1p/

[97] *Hiran, Kamal & Lakhwani, Kamlesh & Wireko, Joseph & Gianey, Hemant*, Internet of Things (IoT): Principles, Paradigms and Applications of IoT, February 2020

[98] *K. Ashton*, That 'Internet of Things' Thing, *RFID Journal*, June 22, 2009

[99] *N. M. Radziwill*, The Fourth Industrial Revolution: Klaus Schwab. 2016. World Economic Forum, Geneva, Switzerland, *Quality Management Journal, 25(2),* 108–109, 2018. doi:10.1080/10686967.2018.1436355

[100] European Commission: Directorate-General for Research and Innovation, Renda, A., Schwaag Serger, S., Tataj, D., Morlet, A. et al, Industry 5.0, a transformative vision for Europe – Governing systemic transformations towards a sustainable industry, *Publications Office of the European Union*, 2021. doi:10.2777/17322

[101] *T. S. Kuhn*, The structure of scientific revolutions 2nd ed., *Chicago, London: University of Chicago Press Ltd*. 210 pages, 1970. doi:10.5897/PPR2013.0102

ОКРЕМІ АСПЕКТИ ЦИФРОВОГО ПРЕДСТАВЛЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

А. Шупарський, Ю. Фургала

Львівський національний університет імені Івана Франка,
вул. Драгоманова, 50, 79005 Львів, Україна
anatoliy.shuparskyy@lnu.edu.ua, yuriy.furhala@lnu.edu.ua

З погляду на еволюцію обчислювальних пристроїв і відповідних задач застосування (use case), стаття структурує хронологію розвитку інформаційних систем від ранніх електромеханічних зразків, електронних вакуумних ламп, транзисторів та інтегральних мікросхем, мікропроцесорів, персональних комп'ютерів, до віддалених обчислень та розподілених систем автономних пристроїв, досліджуючи при цьому різні підходи до цифрового представлення комп'ютерних сутностей. Визначивши сферу дослідження та організувавши наукові джерела в хронологічному порядку, стаття відбирає та виявляє зв'язки між окремими подіями, надає огляд та критичний аналіз, а також висвітлює очікувані розвиток та зміни в підходах до цифрового представлення.

Так, у статті розглядається рух від операцій та операндів, їх подальше ускладнення до коду програм та структур даних, а також перехід від процедурного до структурного та об'єктно-орієнтованого програмування (ООП). Клієнт-серверні рішення, реалізовані за допомогою статичного ООП і систем реляційного управління даними, розглядаються як вершина монолітної архітектури. У подальшому, доменно-орієнтований дизайн (DDD) та архітектура мікросервісів розглядаються як сучасні методи рішень віддалених хмарних обчислень. Далі в статті обговорюється розвиток Інтернету речей (IoT), поява розумних речей і цифрових двійників (digital twins), описуються новаторські задачі застосування у глобальної цифровізації, такі як Індустрія 5.0 (Industy 5.0), і виявляються обмеження існуючих методів для відповідного цифрового представлення.

Зрештою, стаття пропонує до розгляду новітній метод цифрового представлення, що використовує постнекласичну парадигму в комп'ютерних науках. Метод прогресує від наперед визначених типів і структур на перевагу динамічних, заснованих на взаємодії сутностей, чим уможливлює цілісне та адаптивне проектування розподілених систем. Подальші напрямки досліджень передбачають формальну специфікацію цього підходу та розробку інструментів для його реалізації в складних розподілених системах.

Ключові слова: цифрове представлення, архітектура рішень, парадигми програмування, розподілені системи, інтернет речей, цифровий двійник, індустрія 5.0

The article was received by the editorial office on 30.08.2024.
Accepted for publication on 14.09.2024.